NAVSWC TR 91-588

ENGINEERING OF MASSIVELY INTERCONNECTED COMPUTER SYSTEMS

BY MICHAEL JENKINS CHARLES YEH STEVEN HOWELL

UNDERWATER SYSTEMS DEPARTMENT

1 OCTOBER 1991

NAVAL SURFACE WARFARE CENTER

Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

NAVSWC TR 91-588

ENGINEERING OF MASSIVELY INTERCONNECTED COMPUTER SYSTEMS

BY MICHAEL JENKINS CHARLES YEH STEVEN HOWELL
UNDERWATER SYSTEMS DEPARTMENT

1 OCTOBER 1991

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

FOREWORD

This report provides an analysis of the design of communication intensive systems. It presents background information on resources used in such a design, and discusses research conducted in this field as presented in the literature. It also describes the current status of on-going research in this area.

This work was performed at Naval Surface Warfare Center (NAVSWC), under the Massively Interconnected Processor Prototype Task, within the Systems Design Synthesis Technology Project (RS34P11) of the Engineering of Complex Systems (ECS) Technology Program. The work is sponsored by the Office of Naval Technology (Code ONT-22). The goal of the Massively Interconnected Processor Prototype Task is to improve current ad hoc design methods. This will be done through development and augmentation of techniques for designing and implementing communication intensive systems through careful and predictable evaluation of algorithmic and architectural alternatives.

The authors wish to thank their sponsor, the Office of Naval Technology, especially Cmdr. Jane Van Fossen, USN (Ret.) and Elizabeth Wald. We appreciate the efforts of Phil Hwang, Harold Szu, and all who helped to refine the Massively Interconnected Processor Prototype Task. We also thank those who helped to edit this report, especially our technical writer, Adrien Meskin.

Approved by:

C. A. Kalivretenos

C. A. KALIVRETENOS, Deputy Head
Underwater Systems Department

ABSTRACT

This document provides an analysis of design elements whose implementation involves algorithmic tasks and hardware resources which must handle large amounts of data or which otherwise present communication (data flow) impasses to the timely (or functional) fulfillment of a design. It proposes a methodology for evaluating candidate resources and selecting those which reduce communication complexity and meet system requirements. The techniques of this methodology should integrate with other steps in the design process so that necessary information is not ignored or lost. They should also be easily automated, since the number of combinations involved may be intractable if attempted by hand. As an indication of future research, it presents background information on resources used in the design of such systems and addresses the implications of introducing physical constraints on these design elements.

CONTENTS

CONTENTS (Cont.)

CHAPTER 1

INTRODUCTION

This document describes a research effort under the Engineering of Complex Systems (ECS) Technology Block at the Naval Surface Warfare Center (NAVSWC). It provides an analysis of design elements whose implementation involves algorithmic tasks and hardware resources which must handle large amounts of data or which otherwise present communication (data flow) impasses to the timely (or functional) fulfillment of a design. It proposes a methodology for evaluating candidate resources and selecting those which reduce communication complexity and meet system requirements. The techniques of this methodology should integrate with other steps in the design process so that necessary information is not ignored or lost. They should also be easily automated, since the number of combinations involved may be intractable if attempted by hand.

Some sections of this document represent the intended structure of future research in this task. Background information has been provided for design resources and metrics to show the direction of the work, and to provide motivation for the foundation of the first chapters. These sections will be updated and expanded in later versions of this paper.

Chapter 2 addresses the classification of algorithms through characteristics which completely describe their nature and which can be related to implementation strategies. A similar treatment is given for interconnection networks in Chapter 3. These characterizations are used in Chapter 4 to examine trade-offs and evaluate the effectiveness of combinations of implementation pairs. This chapter also explains the metrics used for evaluation. Chapter 5 specifies a mechanism for describing commercial and custom massively interconnected model (MIM) components in a format that is acceptable to a toolset which would aid in simulation and evaluation of implementation alternatives.

1.1 BACKGROUND

Ever increasing demands are made on Navy systems. Examples of these demands include greater functionality, faster processing, more capacity, better reliability and survivability. This is partially a reaction to the environment in which the users find themselves, and to the lure of performance and capability provided by insertion of leading-edge technology.

The development of these systems for use in Navy applications is hindered by two growing forces: complexity in the scope of the system itself and complexity in the technologies which are used to implement it. This applies to both new development thrusts and upgraded plans for older systems.

These complexities cannot be accommodated by standard design techniques. New complexities require new ways of thinking about design components and their integration.

1.2 PROBLEM

No matter how careful the designers intend to be, current methods for producing designs of large, complex systems are one-shot processes. Hardware resources are usually chosen first on the basis of "standard" machines or what is most readily available. This leaves the solution of many performance problems to "clever programmers" who can exploit (often undocumented) aspects of the hardware to meet the required function and throughput. Upgrading functionality or performance may then be impossible, since the original issues of system layout and partitioning were never adequately dealt with; therefore, the effects of new functions or resources would be indeterminate.

One response to the problem of large system complexity has been to design relatively non-complex systems and integrate them. Another response is to use current design methods and resources which employ advanced technology and architectural elements. Either way, the problem of distributed function and control must be dealt with.

During the design process, problems arise when it is discovered that certain functions that are critical to system operation exceed the capability of proposed resources. These problems largely fall into two categories: computation load and communication complexity. Computation load (i.e., how much processing there is to do) is measured by counting the instructions required to execute the solution on a generalized processor. It is addressed by such remedies as novel (and usually more powerful) processor architectures and balancing task loads among several processors. Communication complexity[1] is a measure of the performance overhead imposed on interprocess communication due to the processor interconnection network. This cost can be harder to address, since the symptoms may not show up in the small test cases typically used to shake-down a system.

The sources of communication complexity are various. Some reasons are

* the interconnection network (ICN) of a system (or more likely, a subsystem) is an ill match for the dataflow of the algorithm to be implemented;

* the actual data load overwhelms the network (presumably designed for a lesser load);

* the network is used for a purpose for which it was not originally intended (such as when requirements change after implementation or when an architecture is imposed on the designer for reasons of cost, compatibility, or reliability).

The problem of communication complexity is not solved by faster components. In fact, faster components may actually exacerbate the problem by making more data available when the network is already congested! Most computers in use today are based on the Von Neumann processor architecture or some variant of it. As applications for these computers start to reach the capacity of their processing capabilities, more power is sought from faster technology. A close study of the workings of the processor show that this is not an adequate solution. In particular, serial execution of serially supplied instructions on serially supplied data creates bottlenecks which more powerful components can assuage but not overcome. It becomes clear that new processor architectures are needed.

In addition to processor technology, many new architectures have been proposed for parallel processing.[2] These propose both better communication resource technology and network topology. As in the case of faster processors, this will be at most a quick fix. Most new architectures are created for a specific application; it is not clear that the network may be reused for other applications, or the trade-offs required are so severe that implementation on a serial architecture may yield better results. Also, descriptions of algorithms assume that data is processed as it is produced; delays caused by routing or distance will slow down a system no matter how fast that system can process. An algorithm which displays exponential parallelism will not show promised improvement on a network in which interprocessor communication is linear.

1.3 APPROACH

The approach taken for the work by the Massively Interconnected Processor Prototype (MIPP) Task of the Systems Design Synthesis Technology Project (RS34P11.4) is to develop a comprehensive methodology for the design of massively interconnected systems. This project is part of the Engineering of Complex Systems Technology Block sponsored by the Office of Naval Technology. These techniques used in this methodology will be incorporated into the system development procedures which are being defined as the result of this Block. It will use information supplied by the Requirements Specification and Traceability Task and the Real-Time Systems Design Capture and Analysis Task. The process described by these techniques will complement those of the Design Structure Allocation and Optimization Task, which examines computation load on a processor network at a more abstract system description level. The data generated by use of the techniques of these tasks will be enhanced by the results of the System Modeling Technology Task.

In a more immediate sense, the data generated by use of these techniques will be in a form that can be turned into practical designs (see Figure 1-1). Algorithms will be described so that their implementation will suit the conditions of the particular system design. Realistic architectures will be described so that they may be quickly tested and implemented. This may be in the form of a commercial-off-the-shelf (COTS) machine of the desired architecture, a custom machine built of COTS components, or a machine specified in a hardware description language and completely custom-built from plans that are generated by a silicon compiler.

The first step in these techniques is characterizing both the candidate algorithms for implementation of the desired function, and the resources by which they may be implemented (see Figure 1-2). From this, workable solutions may be determined which are based on requirements and constraints. There have been efforts made to characterize the components of distributed algorithms and resources.[3,4] These include studies of interconnection networks and production of parallelizing compilers. Some efforts are quite complete and well represented in the literature; others are less well represented. In any case, there is no description of a consistent methodology for examining the trade-offs of using certain resources to implement certain algorithms, or determining where the break-even point occurs when deciding to use custom designs over off-the-shelf ones. The approach of this research effort is to pull together the characterizations already done, fill in the gaps, and show how these can be used to accomplish the design of a massively interconnected system.

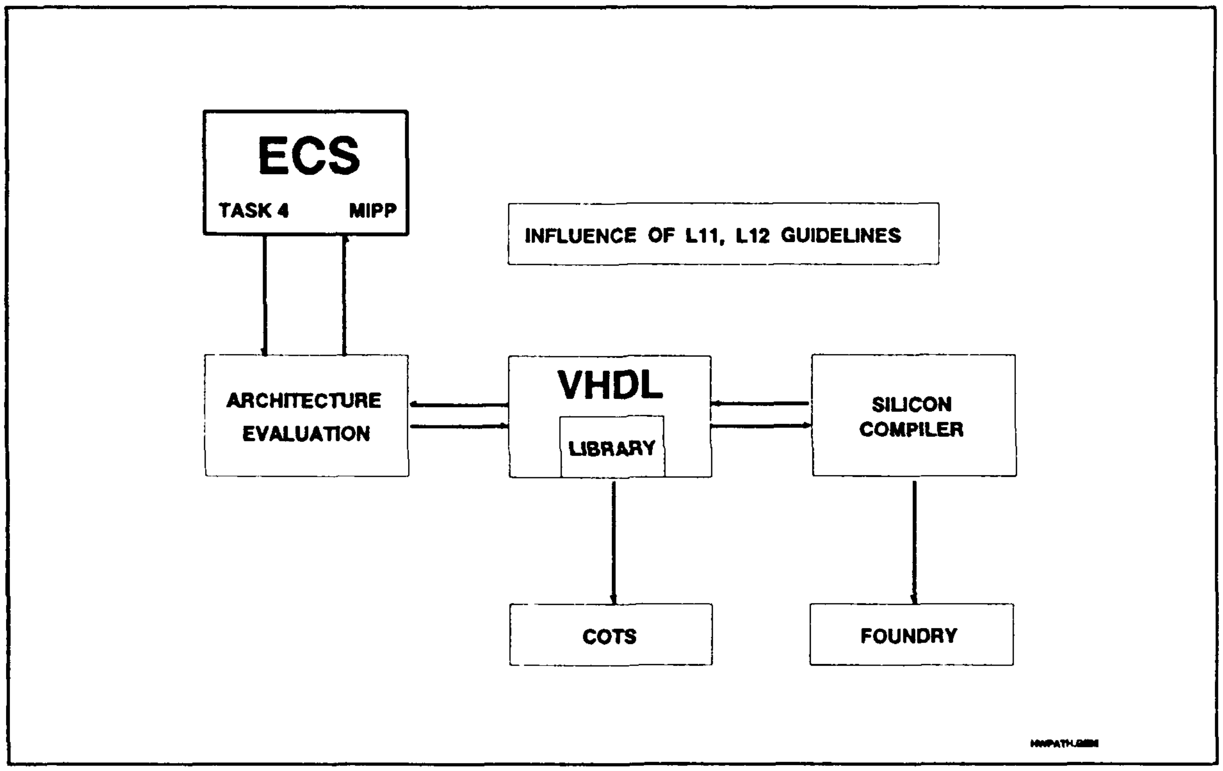
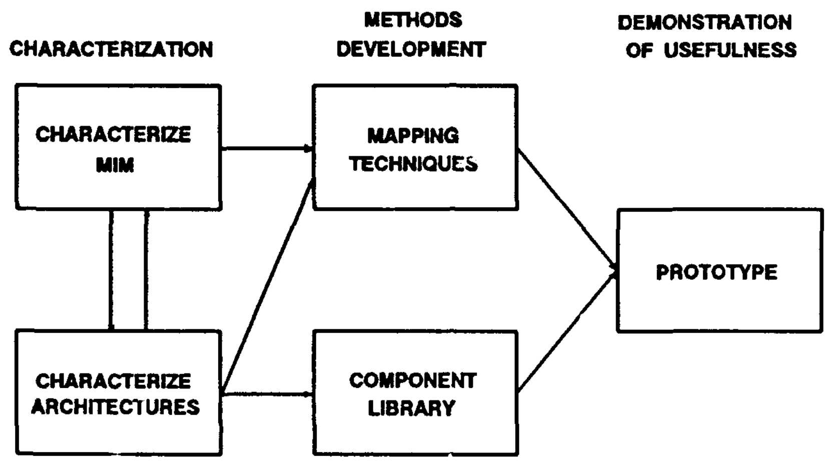


FIGURE 1-1. PATH TO HARDWARE IMPLEMENTATION

This research will result in a method for the development of those portions of systems whose data handling requirements (as opposed to performance requirements) are the driving factor in algorithm selection and resource management. It will enable systems engineers to examine the algorithmic and architectural options for their design, make explicit the trade-offs which are involved in meeting the requirements, and make recommendations in the selection of resources. Attributes of the problem (problem size, data volume, data structure, and throughput requirements) will be used to determine appropriate distributed algorithms and processor network topologies. The latter information will come from a component library (database) of distributed system design paradigms/resources. A mapping can then be done between the algorithm(s) and resource network to determine the effect of communication complexity. The method will produce explicit information concerning the viability of the mapping, alternate design strategies, and issues involved.



DEFINE FORMAL PARAMETERS THAT ALLOW EVALUATION OF MIM COMMUNICATION AND PERFORMANCE PARAMETERS

DEVELOP FORMAL TECHNIQUES THAT OUTLINE A SYSTEMATIC MAPPING FROM ALGORITHMS TO ARCHITECTURES USING COMPONENT LIBRARY

CREATE PROTOTYPE THAT EMBODIES DESIGN TECHNIQUES AND USE RESULTS TO REFINE PROCESS

FIGURE 1-2. APPROACH TO MASSIVELY INTERCONNECTED DESIGN

CHAPTER 2

PARALLEL ALGORITHM CHARACTERIZATION

An algorithm is a rule or a step-by-step procedure for solving a problem. Since this is two levels of abstraction away from implementation (specification and coding follow), it should be easier to manipulate than an interconnection network when trying to find an appropriate match. It also follows that the characterization is more abstract and harder to quantify. Execution of the algorithms selected for solving the problem of massively interconnected systems introduces complexity when required to process high data volumes and rates, or when they have highly interdependent control mechanisms. These constraints dictate the trade-offs that must be made on other non-orthogonal attributes. The identification of these parallel algorithm characteristics and their possible value sets is the first step in creating an intermediate description for comparison to processor topologies. It is then useful to create categories of algorithms so that a relative comparison can be made between similar ones, and new ones can be compared as well. This will also aid in building experience in matching algorithms to networks.

2.1 THE NATURE OF PARALLELISM

Parallelism is the ability of two or more parts of an algorithm to be accomplished simultaneously. It is inherent in the problem to be solved and the tasks required for a solution. Although it can be recognized in an algorithm which has already been specified, exploitation of parallelism can be made easier when parallel implementation is intended from the start. There are two basic types of parallelism that can be recognized in an algorithm, depending on the types of functions to be performed and the data to be manipulated: functional parallelism and data parallelism.

2.1.1 Functional Parallelism

Functional parallelism is found in algorithms which consist of several completely different tasks. The tasks are related to each other by the fact that they contribute to the common function. Completion of one is, however, not necessarily a prerequisite for execution or completion of the other. A simple example is the second degree quadratic equation. For a polynomial of the form $ax^2 + bx + c = 0$, this is stated as:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad (2\text{-}1)$$

Table 2-1 shows the number of time steps required to solve this equation for one root. In the table, t_i is the result of the operation carried out in the ith time step, and $t_{i.j}$ is the jth parallel operation in the ith time step. Since binary arithmetic operators are independent, then all operations for which there are two operands ready can be performed at once (unary operations can always be performed at any time).

TABLE 2-1. TIME STEPS FOR QUADRATIC SOLUTION

Time Step	Sequential Execution	Parallel Execution
t_1	-(b)	-(b), (a)*(c), $(b)^2$, (2)*(a)
t_2	(a) * (c)	(4) * $(t_{1.2})$
t_3	(4) * (t_2)	$(t_{1.3})$ - (t_2)
t_4	$(b)^2$	$\surd$ (t_3)
t_5	(t_4) - (t_3)	$(t_{1.1}) \pm (t_4)$
t_6	$\surd$ (t5)	(t_5) / $(t_{1.4})$
t_7	$(t_1) \pm (t_5)$	
t_8	(2) * (a)	
t_9	(t_7) / (t_8)	

If carried out sequentially, the computation would take nine steps. Four operations are independant and may be carried out on the first time step. If independent functions are performed simultaneously, the result is obtained after only six steps. The difference is the measure of functional parallelism.

These types of tasks are usually independent and exhibit high cohesion. Tasks which are not independent may create problems on a common network. When several tasks operate on the same data set, tokens or keys must be passed. When synchronization is necessary, then control information must be transmitted. Problems arise when this traffic is more cumbersome than the needed processing.

2.1.2 Data Parallelism

Some algorithms operate on large amounts of data, such as those which process sensor data. Frequently, the data can be partitioned into sets, and the same algorithm can be used on each set (see Figure 2-1). The results of these sets then either comprise the answer, or some post-processing can be done on the reduced data to produce the desired outcome. The data set is said to exhibit data parallelism. This type of parallelism is frequently exploited in algorithms which manipulate matrixes, since under many operations they may be independently partitioned. This sort of parallelism does not present problems unless its implementation causes the data to be poorly partitioned, or there are synchronization requirements. The degree of parallelism is determined by the number of independent sets into which the data can be divided.

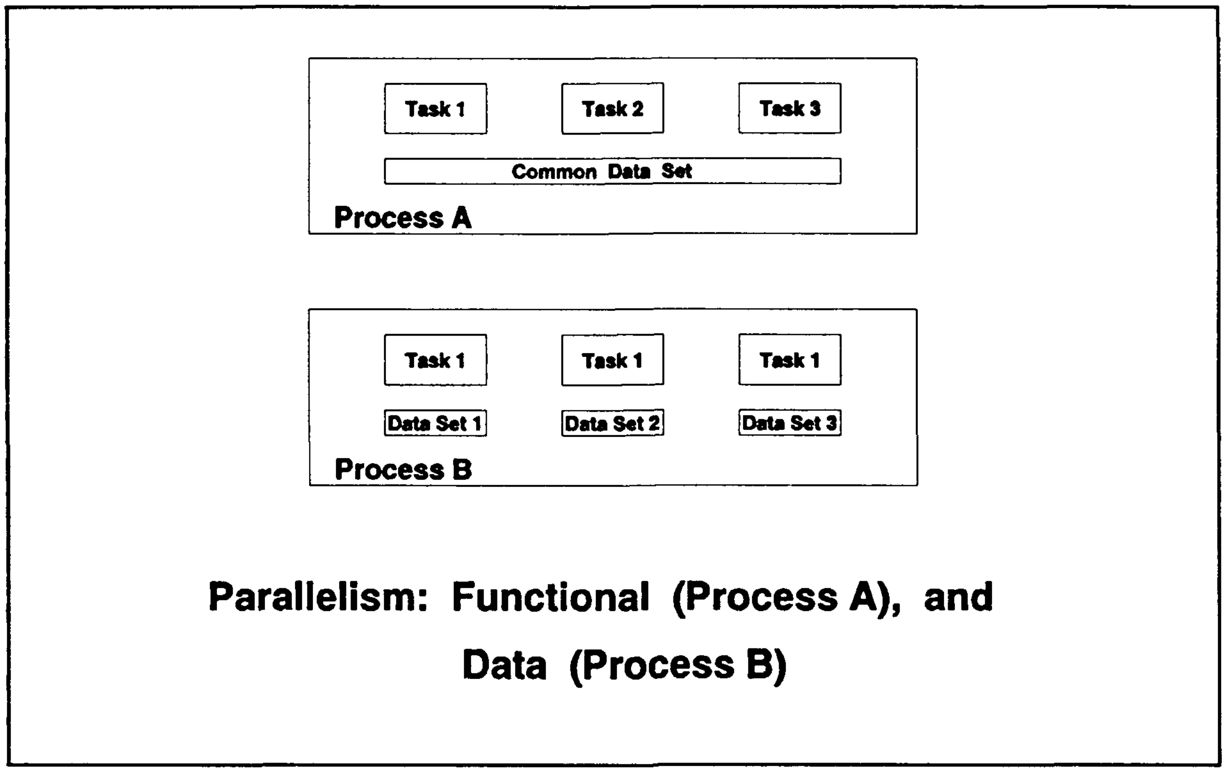


FIGURE 2-1. TYPES OF PARALLELISM

2.2 CHARACTERISTICS OF PARALLEL ALGORITHMS

Parallel algorithms may be loosely characterized according to how they exhibit parallelism. It may appear in many aspects, including the operations involved, the data used, or the control overhead required for process creation and message setup.

2.2.1 Operations

The set of operations to be performed in completing an algorithm depends on the level of abstraction at which development takes place. The operations of the set may be small and related, such as the binary arithmetic operations needed for the quadratic equation above (Equation (2-1)). They may also be large and diverse, such as all of the operations which go into creating a display. The operations required for algorithm development will influence the cohesiveness of the tasks which make up the implementation. (Cohesion is a description of how the tasks in a process are related.) Small operations which are functionally related reduce the number of memory references and require less synchronization. Large operations which contain non-related functions will produce interdependencies and references across data spaces. This attribute, at an instruction level, may determine the regularity of the network implementation. It will have an impact on where data may flow, and which data paths are likely to be overloaded. The smaller and more regular the operations, the more regular and tightly connected the network is likely to be.

2.2.2 <u>Data</u>

The characteristics which have the most impact on communication requirements in a network are those which relate to the data that the network transmits. Data granularity and dependency are two examples of this.

2.2.2.1 <u>Data Granularity</u>. Data granularity refers to the size and uniformity of a data element. A data element is the smallest piece or set of data on which an operation can perform. Data elements which represent one data point (a single scalar) or a set of highly related data points (an array of data representing the value of one point over time) are considered to be of small granularity, since the communication to transport them and the processing to transform them is quick and regular. Data elements which represent a set of loosely or arbitrarily related data points (a "record" or "frame" of data) will require more care in handling. In processing, one or more items of the record may be required, but in general the entire element must be requested and be received in its entirety before it can be transformed. This places performance constraints on the network, which may have to wait for all parts of a large data element to become available to form packets, or maintain a dynamic connection for a long time.

2.2.2.2 <u>Data Dependency</u>. Data dependency, or data coupling, refers to the relation of data elements to each other and to the operations performed on them. They may have a regular arrangement, as in a matrix. They may have a sequential relationship, such as intermediate results, where the results of one operation become the operands for a second. Inputs may be represented by a string of similar data with irregular arrivals, requiring irregular scheduling of the operations to handle them. While low data coupling is generally desirable in software design, dependencies inherent in an algorithm or data structure may be easily handled by an appropriate network architecture. The regularity of the dependency will be reflected in the regularity of the network which can be used to implement the algorithm.

2.2.3 <u>Control</u>

Control information is another component of communication demand on a network. An algorithm may require configuration information or sequencing information to be passed from process to process, or from a master process to subordinate processes. This is due to the method used to distribute and manage execution of the tasks.

On a process level, this characteristic will show up as a pattern of process creations and terminations. If this pattern is static, then it may be determined before implementation and it will be easier to implement the known operations in hardware. If it is dynamic, then creation and scheduling of processes will be another task of the network. This will add control information on the communication channels, which may be broadcast or may have to be transferred to individual processes. Real-time applications will want static patterns, but priority schemes demand dynamic ones.

Control is generally exercised on the algorithm when it is implemented. The way in which the algorithm is specified or implemented may contribute to the communication burden required of the control processes set up by the system and therefore on the network. This attribute is difficult to quantify. In general, it can be reduced by careful partitioning of problems and data sets, and of their allocation to appropriate resources. Indeed, proper selection of tasks and resources is the point of this research.

2.3 TECHNIQUES FOR ALGORITHM CLASSIFICATION

The above characteristics can be used to classify algorithms and put them into groups that will emphasize their similarities. This will make the job of network selection easier by using prior experience with related algorithms. Since an algorithm may be characterized at many stages during its development, classifications exist at many levels. It will help the designer in the long run to identify traits in the earliest stage at which they become apparent. This will positively influence the network architecture selection. There are many perspectives from which to view the development process, and each will provide insights into the nature of the algorithm. It is not important that the views (and characterizations thereof) look the same; in fact, differences help to examine the nature of the problem, and thus explore the solution space.

2.3.1 Characterization of Existing Algorithms: Application Domain Grouping

There are, of course, many algorithms already documented. It may be preferrable to use one which is already specified to save time or maintain compatibility. If not specified, one way to begin characterization of an algorithm is to examine the application domain into which it falls. The application domain is defined by similarity of function and operating environment.

2.3.1.1 Digital Signal Processing. Algorithms in this area tend to have functional parallelism, process level operations, data dependency and coarse data granularity. The signals for processing usually originate from different sensor platforms, and thus require different sets of process level operations for processing. Examples of these algorithms are digital filters and fast Fourier transforms.

2.3.1.2 Image Processing. Algorithms tend to have data parallelism, instruction level operations, low data dependency and fine data granularity. The images to be processed are decomposed into many small regions of variable size. Uniform operations are applied to each pixel with the result used as input to immediate neighbors. An example is edge detection.

2.3.1.3 Symbolic Processing. These algorithms tend to be characterized by data parallelism, process level operations, high data dependency, and medium data granularity. Symbolic processing is required by most artificial intelligence applications, such as pattern matching.

2.3.1.4 Graphics Processing. This domain is complex, including both manipulation of geometric objects and translation to a display image.

Geometric manipulation algorithms tend to have functional parallelism, process level operations, data dependency and coarse data granularity. Examples are scaling, rotation, and translation of objects.

2.3.2 Characterization of Algorithms in Development

Characteristics of algorithms can be defined at stages in the development process. Some characteristics will be determined by the method used for development; others may need to be decided on in order to proceed. Development starts with a plan for attacking a problem, which leads to a logical model of the algorithm. Finally, the algorithm is formally specified, and consequentially implemented.

2.3.2.1 Logical Model of Algorithm. A logical model of an algorithm is the result of an analysis of the function to be performed. It is the formal approach to the solution of a problem.

2.3.2.1.1 SEQUENTIAL MODEL. A sequential model of approach is appropriate when there is no more than one thing to do per time step. It may also be imposed by a hardware constraint (i.e., only one processor may be allocated to the algorithm due to cost or hardware failure). While this obviates the need for an interconnection network (and thus our consideration), it is useful to note the costs of uniprocessor execution during development of a parallel algorithm. This gives a baseline for measuring the improvement of the parallel system. It also helps in estimating the effects of hardware failure.

2.3.2.1.2 PARALLEL MODEL. A parallel model of approach is appropriate when, at any time step, more than one thing can be accomplished. Note that since time dependencies always exist, elements of the sequential case are included here.

Sequential Requirements. It is occasionally necessary to impose restrictions on the sequence of the performance of functions that may otherwise be performed in parallel. This may be in order to ensure "correct" operation or to fulfill a higher level requirement, such as safety. For example, a missile may be armed at the same time it is fired. In the interests of avoiding damage to the ship and crew, it may be desirable to delay arming until the missile is well clear of the ship.

Divide-and-Conquer. Divide-and-conquer approaches address those problems for which the data set or process set (task) can be broken up into several fairly independent portions. The communication in this case can be of two types. The first is pre- and post-processing management of processes. This is central control information from a master processor, or handshaking information from processors with distributed control. The second is the transmission of intermediate answers (perhaps as operands for a higher level function) or the aggregation of results to a central processor. This type of approach is considered massively interconnected if control requirements are numerous, or if the computations are small and the data set is large. In such an environment, it is possible for the synchronization of the management and aggregation of data to consume more time than that of the processing.

Systolic. Systolic (and pipelined) approaches are relevant to a specific class of problems with rigidly defined instruction and data relationships. In these problems, the transformations performed on data are simple, few in number, and can be applied rhythmically. The data flow itself is directed; that is, the data structure is regular and elements of that structure are used in a regular sequence of operations. The symmetry of this approach is so rigid that a network of hexagonally interconnected processors is called a "systolic array." This type of approach is also suitable for arrays of other degree (connections per node) and dimension.

2.3.3 Algorithm Specification

Once a model of the algorithm has been constructed, it must be specified in some formal manner such that it may be implemented. There are several means for this specification, and each will reveal (or impose) characteristics of the final implementation.

2.3.3.1 Sequential Specification. An algorithm may be specified by a standard sequential specification method. A ubiquitous example of this is the flowchart. Others include Nassi-Shneiderman diagrams and program description language (PDL). These are the specification methods that are used for current sequential high-level language compilers. They match the sequential flow of the languages so well that elements of the intended coding language are often used (generally recognized as bad design practice). These are usually implemented using a library of functions prepared for the hardware. The parallelism may not be obvious and frequently depends on the libraries used. While these expressions may be logically rigorous, they do not lend themselves to parallel computer implementation.

2.3.3.2 Functional Language. A functional language is used to express the object transformations necessary to describe an application. It consists of objects, definitions, and functions. Functions are defined and then applied to objects to create new objects, or combined to create complex functions. Sequence is not implied by order of expressions. Control constructs are not used. This ensures that any expressions whose operands are known may be resolved simultaneously.

2.3.3.3 Vector Notation. Algorithms may be expressed in a mathematical notation where the operands represent vector quantities. These expressions represent data which may be processed in parallel. In addition, expressions which are not interdependent may also be resolved simultaneously.

CHAPTER 3

COMMUNICATION NETWORK CHARACTERIZATION

In order to accurately assess the usefulness of a particular interconnection network for a given application, it is necessary to precisely describe the network. The description should focus on those aspects of the network which address the problem at hand, in this case, the problem of communication intensive systems. To this end, the general characteristics of interconnections will be explored, and a useful taxonomy of architectures will be described that will help the designer to make educated predictions about the network's usefulness for a particular type of application.

3.1 CHARACTERISTICS OF INTERCONNECTION NETWORKS

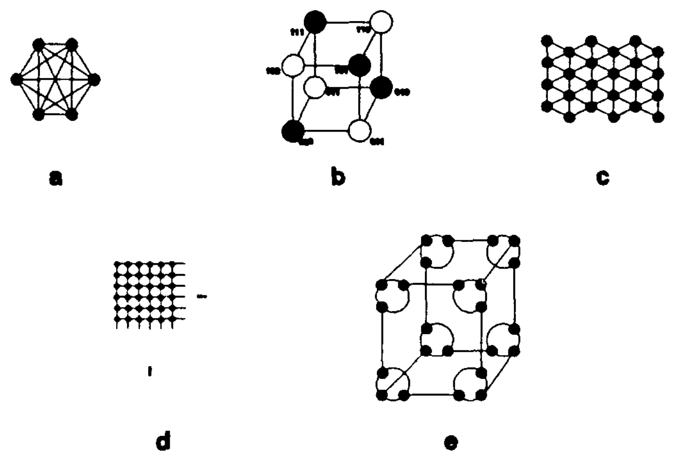
Many architectures have been proposed for division of the computation involved in the execution of parallel algorithms. Some are general groupings of processors with communication paths to collect results; others are carefully thought out arrangements whose data paths match that of the algorithm for which they were specifically designed. The following represents characteristics of these architectures and investigates their usefulness for communication intensive applications.

3.1.1 Node Address

When discussing communication in interconnection networks, the capability of the processor is usually considered an external property. Only those features of the processor that describe the interface to the network (e.g., the amount and rate of data pumped into the system) are considered. The processors can be replaced by a node address that will uniquely identify its position in the network. These addresses are often expressed in binary form, with each digit representing one connection to the node (in a multi-dimensional system). Then, any communication on the network is a transformation of addresses, as shown in the section on interconnection functions (Section 3.1.3).

3.1.2 Interconnection Type

The interconnections between nodes can be either static or dynamic. A connection is static if the node addresses at each end are always the same (see Figure 3-1). A node with static connections may only directly communicate to those nodes at the other end of the connection; information meant for other nodes must be routed through these. The extreme example of this is a fully interconnected network; i.e., a one-to-one connection between each pair of nodes. The node addresses in this case need never change, since the connection is asked to service only one set of nodes.



Static Network Topologies: (a) Fully Connected, (b) Hypercube, (c) Systolic Array, (d) 2-D Mesh, (e) Cube-connected Cycles

FIGURE 3-1. ARCHITECTURES WITH STATIC INTERCONNECTIONS

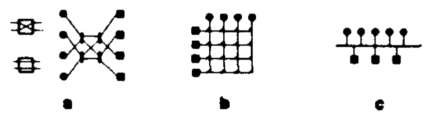
A connection is dynamic if the addresses at each end can change over time (see Figure 3-2). This is accomplished through a switching element. The switching element can connect any one of its inputs to any one of its outputs at one time. The switches may be set by a controller before communication begins, or this control may be distributed so that each switch examines the information passing through it to determine a proper setting. The extreme example of this connection is a bus. Only one connection is supplied for the entire set of nodes. Two addresses are selected (through proper setting of switches) for communication on the connection at any one time; the others must wait.

3.1.3 Interconnection Functions/Routing Algorithms

In studying interconnection networks, it is useful to think of the network as "processing" the information traffic, as the processor processes the data. The network imposes a function on the traffic passing through it. An interconnection function[5] is a transformation on a node address which results in the address of one other node to which information can be directly sent (i.e., in one pass through the network). A number of interconnection functions may be required to completely describe all of the direct connections of a network. For example, the interconnection function for a binary cube is

$$cube_i\ [b_{d-1} \ldots b_i \ldots b_0] = [b_{d-1} \ldots \overline{b_i} \ldots b_0]$$

where $[b_{d-1} \ldots b_i \ldots b_0]$ is a node address. This says that a node in a binary cube is directly connected to any node whose address differs by one binary bit. The connections of a binary cube are completely described by N $cube_i$ functions, where N is the number of dimensions.



Dynamic Network Topologies: (a) Switching Network with exploded view of switching element (both states), (b) Crossbar Switch Network, (c) Buss Connected Processors

FIGURE 3-2. ARCHITECTURES WITH DYNAMIC INTERCONNECTIONS

A routing algorithm is a set of the steps that indicate how information is to pass from one node to another in a network. It is usually an operation on the terminal node addresses, regardless of any intervening nodes. The routing algorithm for the binary cube is as follows:

> To send data from node X to node Y, perform a logical XOR on the addresses. The data must be passed once in each direction (dimension) for which there is a 1 in the result. No required order of passes is implied by bit position.

Notice that this routing algorithm is accomplished by repeated application of the interconnection function, once for each bit position containing a 1.

3.1.4 Contention/Arbitration Schemes

A network which provides less than one-to-one bidirectional connections for each pair of nodes has the potential for contention. The method used to arbitrate this contention may have an effect on the way information is processed by the system overall. There are several ways of dealing with contention. If the effects are known and manageable, then contention may be allowed to exist. An example of this is a situation in which there are two sets of sending and receiving node pairs, one at each extreme end of a long backplane (bus); the transmissions are complete before the wavefronts which carry the information collide. If this is not acceptable (as is usually the case), traffic must be explicitly regulated.

The most common arbitration schemes are based on token passing. In a token passing scheme, permission to use communication links is granted to one node. That node may set up a circuit to another node, put a packet of messages on the network, or broadcast information to all nodes, depending on the communication protocol. The token is then passed to another node. If the scheme is first come, first served, the next node which has made a request will get the token. Another scheme is round-robin, where the next node in a predetermined line-up (perhaps by node address) will receive the token, passing it on immediately if it has no communication to transact. While the scheme tries to make sure that all messages are passed in a reasonable amount of time, it is not always desirable to be fair. Some applications (such as real-time) require that communication paths be open between certain nodes regardless of the needs of others. Note that these schemes are automatic; that is, the token is passed based on a queue of which all nodes can be made aware. This allows for more distributed control and requires no synchronization of the network.

If some communication is identified as critical, then the arbitration scheme must allow for preemption. For instance, data from all sensors will be processed while surveilling. When a detection is suspected, only data from that direction and frequency range will be of interest. Communication of other data will be of secondary importance. A controlling process is needed to watch for such conditions, suspend other processes, transfer complete use of the network to the critical process, and determine when the critical time has passed or if any other communication is to be allowed during the critical period.

3.1.5 Bandwidth

Bandwidth is a measure of capacity. A network's bandwidth is the number of transmissions it can perform at once. More connections mean more bandwidth. The bandwidth of a less than fully connected network will depend heavily on both the protocol used to send information, and the nature of the information. A reconfigurable network is one which allows an existing communication circuit to be rerouted if another request for a connection is received, and it is determined that there is a configuration which will accommodate both requests. This network (both the topology and protocol) will perform better than one which forces subsequent requests to wait. On the other hand, long "conversations" between nodes are more efficient if not constantly interrupted. A network which bundles many short messages together and maintains a connection is likely to have a higher bandwidth than one which is constantly trying to make and break connections.

The bandwidth of each connection is also of interest and will influence the network bandwidth. The physics of the implementation technology will impose limits on the rate at which data can be transmitted. Typical coaxial cable data rates are 10 megabits per second; fiber optic cables give ten times that performance. While this is irrelevant to simultaneous requests, it is extremely important in that the network spends less time in a busy state while nodes (processors and memories of ever increasing speed and capacity) create higher demands.

3.1.6 Emulation

Emulation is the ability of one network to perform the function of another. This is usually done through a repeated application of the interconnection function of the first in a particular sequence that accomplishes the interconnection function of the second. For example, the function of a linear network

$$lin\ B_i = \begin{array}{ll} B_{i+1}, & i \neq 0 \\ B_0, & i = 1 \end{array}$$

can be emulated by the binary cube function by applying it to bit positions starting from the least significant, up to and including the first position which contains a zero.

Emulation is best when the desired interconnection function can be realized at no additional cost. If a direct connection on the desired topology is emulated through a connection which contains a node (and possibly a store-and-forward) on the actual machine, then additional costs are incurred. These costs must be quantified and analyzed to determine whether they are acceptable for the required response.

An example of the importance of the ability of one network to emulate another is when one network must be used to implement two applications with dissimilar data flow requirements. This situation may arise through constraints of cost or space. If one application's data flow does not match the actual network topology, then the actual network may be required to emulate another, more suitable topology.

3.1.7 Partitionability/Scalability

It may be useful to divide a network into subnetworks of the same topology. Examples of a need for this are logical task partitioning or multiple user demands. This is not a problem for mesh architectures, where node addresses can be reassigned, and interconnection functions applied so as to only transmit data within the subnet. It might present problems for a switching network, where the ability to create a set of connections may depend on the whole network being available at a given time. To ensure partitionability, the designer must be sure that the routing algorithm of the whole applies to the part, and that each node in the subnet can be reached by proper application of the interconnection function to another node in the subnet.

It is useful and often necessary at other times to enlarge networks through the addition of nodes and connections according to the existing interconnection scheme. For adding nodes to an array, this is no problem. Since the degree of connection is two, and each node is connected only to the preceding and following node, both physical and communication complexity are kept to a minimum. For more exotic architectures, scaling up may mean the addition of whole dimensions (enlargement of a 64 node binary cube by one dimension requires another 64 nodes and 448 extra connections!).

3.1.8 Fault Tolerance

Designers who seek to take advantage of distributed processing networks expect not only increased performance, but that critical tasks can continue to be processed in the event of node failure. This also applies to the interconnections. It is important to know what the impact will be should a connection cease to function for whatever reason. For each node, this includes a list of reachable nodes, an estimate of increased transmission time, and reconfiguration information.

3.2 PARALLEL ARCHITECTURE TAXONOMIES

Several taxonomies of parallel architectures are documented in the literature. These are created either to serve as an academic aid or to assist researchers in classifying new architectures and recognizing similarities/differences in the groups. A taxonomy will be used in this work to help make explicit characteristic considerations and to ensure that the complete field of architectures (the architectural design space) is covered. It will reflect previous work so that consistency with current thinking may be maintained.

3.2.1 Flynn's Taxonomy

The taxonomy most widely used in computer literature was described by Michael Flynn in 1972.[6] It classifies computer architectures according to the number of instruction and data streams that are simultaneously present. Specifically, it defines four classes:

Single Instruction stream, Single Data stream (SISD):
This class is the omnipresent uniprocessor Von Neumann machine. Examples are the Intel 80486 and Motorola 68040. Although crucial to so many (now) everyday tasks, it is the "Von Neumann bottleneck" (sequential, single-step instruction and memory fetching) that parallel architectures seek to overcome.

Single Instruction stream, Multiple Data stream (SIMD):
In this category, all processors in the system execute the same instruction at the same time on different data. One variation is the ability to disable some processors, or partition a large network into several smaller ones. Examples are Cray computers (pipelined vector processors) and the Warp machine (systolic array).

Multiple Instruction stream, Single Data stream (MISD):
Although some functional languages would support this type of execution with the ability to perform several functions on the same datum, there are currently no architectures of this type extant; it is generally thought to be not useful.

Multiple Instruction stream, Multiple Data stream (MIMD):
This class includes loosely- and tightly-coupled multiprocessor networks. Each processor executes its own set of instructions on its own data set. Each processor executes according to its own clock, and data is passed either through memory or buffers. Examples might be hypercubes and meshes.

While this taxonomy is quite useful in general discussion, it is very broad and does not address the growing number of hybrid and singular architectures, nor can it consider new distinctions between, and uses for, old topologies.

3.2.2 Feng's Classification

In 1981, Tse-yun Feng produced a survey of interconnection networks.[7] His description was based on four properties or elements. Each element had two major values for classification. The elements were operation mode, control strategy, switching method, and network topology. These are described below.

Operation mode of communication can be either synchronous or asynchronous. The operation mode depends largely on the type of data and type of processing to be done by the network. In a message-passing binary cube, where independent tasks are carried out by the nodes, communication will be asynchronous. In a systolic array, where the direction and timing of the information flow identifies its purpose (for example, whether it is an input, intermediate, or output value), communication is synchronous.

Control strategy describes the management of switching elements (or package routing). This can be controlled through a central management scheme, from which binary masks are applied to determine status (crossed or not) of switches, or communication is uni-directional and lock-step (as in a systolic array). Alternatively, distributed control allows each switch (or connection) to determine its disposition as it passes the information at its input. Important characteristics are interconnection function and partitionability/scalability.

Switching method describes the nature of data transmission. This can either be done by establishing a permanent or non-permanent physical path prior to data stream transmission (circuit switching), or by bundling messages together and including routing information in a header so that the path may be determined en route (packet switching). This reflects the interconnection type, routing algorithm, and bandwidth characteristics.

Network topology describes the internodal connections. Direct connections between nodes are either static, or can be built and broken. The first level classification of this element is identical to the Interconnection Type characteristic (Section 3.1.2). Lower level classifications include spacial dimension (i.e., one or three dimensional meshes), layers (i.e., one or three layer switching networks), and hybrids. This element is often best described by a graph or formula, such as those in Figures 3-1 and 3-2.

This survey provides the most useful classification for this discussion, because it uses the connections to differentiate the architectures rather than the positions and types of processors. Some of the distinctions are not inherent to the architecture of the network, but can be selected according to need. For instance, a two dimensional mesh network may be run in synchronous or asynchronous mode.

3.2.3 Other Taxonomies/Surveys

Since Flynn published his taxonomy, many new ideas for multiprocessor and multicomputer architectures have arisen. Consequently, there have been several attempts at general classifications which claim to include these new architectures.[2,8,9] These classify architectures based on differences in the processor type, memory allocation scheme, and data flow imposed by the system. Most refer to both Flynn's and Feng's work. Skillicorn's classification in 1988[9] was quite comprehensive in an attempt to define species from original and unique criteria while remaining consistent with Flynn. It included a description of how the processors might be connected (i.e., 1-1, 1-n, n-n). The major implication was which processors might converse with which others, not about the connections themselves.

These classifications are useful, but most do not address such issues as communication intensity, communication contention (in terms of appropriate protocols, or scalability. Feng's elements show the most promise of being able to effectively describe these network attributes.

CHAPTER 4

IMPLEMENTATION TRADE-OFF

When a system is ready for implementation issues to be introduced, constraints are made on the number and types of resources which are available for realization. These include timing, weight, space, heat, cost, reliability, and standards. These constraints are not independent, and may interact to create an implementation environment which is quite restrictive. The designer will find it necessary to trade off advantages of one design attribute for another. Three situations will be considered here: fixed network and variable algorithm; fixed algorithm and variable network; and both variable network and algorithm (see Figure 4-1).

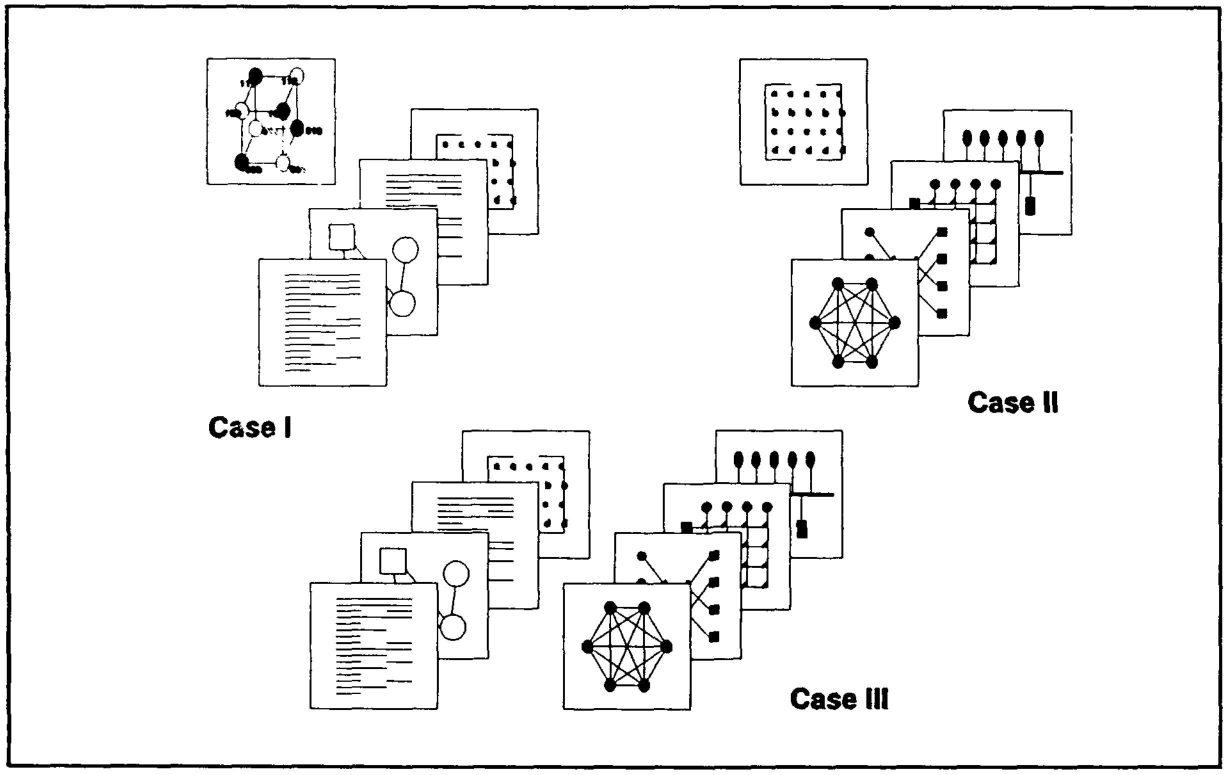


FIGURE 4-1. CHOICE LIMITATIONS AFFECT TRADE-OFFS

4.1 CASE I: FIXED NETWORK AND VARIABLE ALGORITHM

Often, in the Navy and the commercial sector, the type of network to be used in the development of a system is imposed by others. This may occur due to non-functional requirements, for reasons of prior practice, convenience, savings, or for other reasons. This situation is likely to lead to a non-optimal design, and may or may not make the job easier, depending on the flexibility of the imposed network and the diversity of algorithms to be implemented.

Since the network is given, it may be precisely described according to the characteristics in Chapter 3 of this report. The algorithms which are suitable for the desired function and the given network can then be identified and their characteristics examined. These characteristics will be used to identify feasible, and eventually preferable, implementation alternatives. Particular attention should be paid at this stage to the need for network emulation. Simulation and other comparison techniques are used to evaluate the combinations and make clear the trade-offs involved. The techniques for evaluation of this case are the subject of further work in this research effort.

4.2 CASE II: FIXED ALGORITHM AND VARIABLE NETWORK

This situation is less common, usually arising from more natural causes than the previous case. It may happen that there is only one acceptable algorithm which will produce the desired result.

This situation is similar to Case I but reversed. There should be no problem with network emulation unless there is no network suited to the algorithm in the first place. The techniques for evaluation of this case are the subject of further work in this research effort.

4.3 CASE III: BOTH VARIABLE NETWORK AND ALGORITHM

This is the most desirable situation, although it is the most complicated of the three. This case gives the designer the flexibility (and also the responsibility) to examine all possible combinations of networks and algorithms which will produce the desired function according to specifications. The ability to automate the techniques described herein will permit good combinations to be identified so that they may be examined in a timely manner.

Any network or algorithm to be considered for implementation should be characterized as described in previous chapters. This will allow simulation and other experimentation to be performed to examine trade-offs and select an optimal solution. Note that there may be more than one equally desirable solution, or that the best solution (according to some criterium) may not be the preferred one for various reasons. The techniques for evaluation of this case are the subject of further work in this research effort.

4.4 METRICS

The techniques used for evaluation of the above cases will be based on metrics which provide a quantitative basis for decision-making. Many metrics have been used to measure the computational performance of multi-processor computer systems. They are based on a model of a processor network, denoted by the following traits:

p:	number of processors or network nodes
T_p:	time to execute on p processors
n:	size of problem (usu. order of magnitude of data)

The metrics used are[10]

SPEEDUP ($S_p = T_1 / T_p$);
EFFICIENCY ($E_p = S_p / p$); and
USEFUL PROCESS POINT (U_p = smallest n so that $T_p <= T_{p-1}$).

Similar metrics can be found for evaluating the implementation of a certain algorithm on a certain network according to the characteristics of each and the dynamics of the pair. The enumeration and description of these metrics are the subject of further work in this research effort.

CHAPTER 5

MASSIVELY INTERCONNECTED MODEL DESIGN ELEMENTS

The implementation design of a network is accomplished through a selection of components which fulfill the required functions and behaviors. As older technologies are improved and new technologies become available, it is important to keep track of specifications of these components so that the entire range of options for performance and compatibility is kept open. This can be done through a library of elements which encapsulates the functional and behavioral information in an interface (see Figure 1-2). The interface can be used to examine the relationships between components as they would in an actual implementation.

5.1 PROCESSING ELEMENTS

The processor is an external entity to the study of an interconnection network. There are, however, qualities of the processor's interface to the network which are important to the network designer. Some of these are the amount and rate of data put on the network, the types of control that a processor may exercise over the network, and delays and loading associated with the creation, transmission, and reception of messages.

5.2 MEMORY MODEL

Processor networks are generally arranged with two types of memory models: shared or distributed. Like the processors themselves, the memory modules are considered external entities and are not included in consideration of the interconnection network. The type of memory will influence the type of communication introduced to the system and the amount of contention that is likely. When data is passed from one processor to another, distributed memory models demand that a line of communication be open between the processors; i.e., they are simultaneously active. Shared memory models, on the other hand, pass data through memory. If both processors are simultaneously active, blocking will occur. Since blocking affects accessibility of the communication links, this is of concern to the designer.

5.3 INTERCONNECTION NETWORKS

The interconnection network is the most interesting MIM design element, since it (with the communication protocol) determines the function and performance of the communication system. Two basic types of networks are mentioned, fully connected and statically connected.

5.3.1 Fully Connected

Basic elements of fully connected networks (besides point-to-point connections) are the buss and the crossbar switch.

5.3.2 Statically Connected

The basic element of a statically connected network is the connection itself. The elements may differ in terms of technological capability. Also, many static configurations are regular; therefore compound elements may be built up (i.e., an n-by-n section of a mesh).

5.3.3 Dynamically Connected

The basic element of a dynamically connected network is the switching element.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

The intent of this research is to provide systems designers with a set of techniques which will allow them to evaluate the usefulness of a particular massively interconnected algorithm on a particular interconnection network, or to design a new one if necessary. The specific work described in this interim report is the characterization of these algorithms and networks to determine the attributes which will be useful in this evaluation. The attributes found have been discussed in terms of the stress put on a network, on the one hand, and the capacity and capability provided on the other.

The attributes shown can be compared in such a way as to produce quantitative and qualitative metrics. These metrics will account for the influence of constraints and requirements, and will allow the designer to determine an efficient (if not optimal) solution.

REFERENCES

1. Hossfeld, F. "Parallel Algorithms: The Impact of Communication Complexity," Colloquia Mathematica Societatis, Janos Bolyai, 1984.

2. Stone, H., High-Performance Computer Architecture, Addison-Wesley, Reading, MA, 1987.

3. Jamieson, L.H., et al., The Characteristics of Parallel Algorithms, The MIT Press, Cambridge, MA, 1987.

4. Skillicorn, D.B., "Architecture-Independent Parallel Computation," IEEE Computer, Vol. 23, No. 12, Dec 1990, pp. 38-51.

5. Siegel, H.J., Interconnection Networks for Large-Scale Parallel Processing, D.C. Heath and Co., 1985.

6. Flynn, M., "Some Computer Organizations and Their Effectiveness," IEEE Transactions on Computers, C-21, No. 9, Sep 1972, pp. 948-960.

7. Feng, T., "A Survey of Interconnection Networks," IEEE Computer, Dec 1981, pp. 12-27.

8. Duncan, R., "A Survey of Parallel Computer Architectures," IEEE Computer, Feb 1990, pp. 5-16.

9. Skillicorn, D.B., "A Taxonomy for Computer Architectures," IEEE Computer, Vol. 21, No. 11, Dec 1988, pp. 46-57.

10. Finkel, R.A., "Large-Grain Parallelism - Three Case Studies," The Charateristics of Parallel Algorithms, Jamieson, L., Ed., MIT Press, 1987.

BIBLIOGRAPHY

Almasi, G.D. and Gottlieb, A., Highly Parallel Computing, Benjamin/Cummings Publishing Co., 1989.

Bell, G., "The Future of High Performance Computers in Science and Engineering," Comm. ACM, Vol. 32, No. 9, Sep 1989, pp. 1,091-1,101.

Bertsekas, D. and Tsitsiklis, J.N., Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, NJ, 1989.

Davis,L., Handbook of Genetic Algorithm, Van Nostrand Reinhold, 1991.

Denning, P.J. and Tichy, W.F., "Highly Parallel Computation," Science, Nov 1990, pp. 1217-1222.

Duncan, R., "A Survey of Parallel Computer Architectures," IEEE Computer, Feb 1990, pp. 5-16.

Feng, T., "A Survey of Interconnection Networks," IEEE Computer, Dec 1981, pp. 12-27.

Finkel, R.A., "Large-Grain Parallelism - Three Case Studies," The Charateristics of Parallel Algorithms, Jamieson, L., Ed., MIT Press, 1987.

Flynn, M., "Some Computer Organizations and Their Effectiveness," IEEE Transactions on Computers, C-21, No. 9, Sep 1972, pp. 948-960.

Germain, C., et al., "An Interconnection Network and a Routing Scheme for a Massively Parallel Message-Passing Multicomputer," The 3rd Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Soc., Oct 1990, pp. 368-371.

Gibbons, A. and Rytter, W., Efficient Parallel Algorithms, Cambridge University Press, 1988.

Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

Hack, J.J., "On the Promise of General-Purpose Parallel Computing," Parallel Computing, Vol. 10, No. 3, 1989, pp. 273.

Hearne, J. and Jusak, D., "How to Use Up Processors," The 3rd Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Soc., Oct 1990, pp. 515-518.

Hoare, C.A.R., Communicating Sequential Processes, Prentice-Hall, 1985.

Hossfeld, F. "Parallel Algorithms: The Impact of Communication Complexity," Colloquia Mathematica Societatis, Janos Bolyai, 1984.

Hwang, K. and Briggs, F.A., Computer Architecture and Parallel Processing, McGraw-Hill, Hightstown, NJ, 1984.

Jamieson, L.H., et al., The Characteristics of Parallel Algorithms, The MIT Press, Cambridge, MA, 1987.

Karp, A., "Programming for Parallelism," IEEE Computer, Vol. 20, No. 5, May 1987, pp. 43-57.

Kirkpatrick, S., et al. "Optimization by Simulated Annealing," Science, Vol. 220, 1983, pp. 671-680.

Lipovski, G. and Malek, M., Parallel Computing, John Wiley & Sons, 1987.

Mou, Z.G., "Divacon: A Parallel Language for Scientific Computing Based on Divide-and-Conquer," The 3rd Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Soc., Oct 1990, pp. 451-461.

Pancake, C.M. and Bergmark, D.B., "Do Parallel Languages Respond to the Needs of Scientific Programmers?," IEEE Computer, Vol. 23, No. 12, Dec 1990, pp. 13-23.

Reed, D.A. and Fujimoto, R.M., Multicomputer Networks: Message-Based Parallel Processing, The MIT Press, Cambridge, MA, 1987.

Siegel, H.J., Interconnection Networks for Large-Scale Parallel Processing, D.C. Heath and Co., 1985.

Skillicorn, D.B., "A Taxonomy for Computer Architectures," IEEE Computer, Vol. 21, No. 11, Dec 1988, pp. 46-57.

Skillicorn, D.B., "Architecture-Independent Parallel Computation," IEEE Computer, Vol. 23, No. 12, Dec 1990, pp. 38-51.

Soucek, B. and Soucek, M., Neural and Massively Parallel Computers: The Sixth Generation, John Wiley & Sons, New York, NY, 1988.

Stone, H., High-Performance Computer Architecture, Addison-Wesley, Reading, MA, 1987.

Ward, S.A. and Halstead, R.H., Computation Structures, The MIT Press, Cambridge, MA, 1990.

DISTRIBUTION

	Copies
Defense Technical	
Information Center	
Cameron Station	
Alexandria, VA 22314	12
Library of Congress	
Attn: Gift and Exchange Division	4
Washington, DC 20540	
Office of Naval Technology	
Attn: Code 227	
(Elizabeth Wald)	1
(Cmdr. Gracie Thompson)	1
800 N. Quincy Street	
Arlington, VA 22217-5000	
Center for Naval Analyses	
4401 Ford Avenue	
P.O. Box 16268	
Alexandria, VA 22302-0268	2
Naval Postgraduate School	
Attn: Code EC/LE	
(Chin-Hwa Lee)	1
Monterey, CA 93943	
Research Triangle Institute	
Attn: Geoffrey Frank	1
P.O. Box 12194	
Research Triangle Park, NC 27709-2194	
Advanced Technology & Research Corp.	
Attn: Adrien J. Meskin	5
George Stathopoulos	1
14900 Sweitzer Lane	
Laurel, MD 20707	

	Copies
Internal Distribution:	
D4	1
E231	2
E232	3
E342 (GIDEP)	1
F01	1
G07 (F. Moore)	1
G42 (C. Yeh)	1
K02	1
K14 (D. Clark)	1
K52 (W. Farr)	1
N15 (M. Wilson)	1
N30 (H. Crisp)	10
N35 (M. Masters)	1
N35 (F. Riedl)	1
R44 (E. Cohen)	1
R44 (H. Szu)	1
U	1
U02	1
U042	1
U10	1
U20	1
U23 (W. Dence)	1
U23 (J. Horner)	1
U23 (P. Winters)	1
U25	1
U25 (D. Bergstein)	1
U25 (E. Hein)	1
U30	1
U31 (R. Scalzo)	1
U33	1
U302 (P. Hwang)	20
U33 (D. Choi)	1
U33 (M. Edwards)	1
U33 (N. Hoang)	10
U33 (S. Howell)	10
U33 (M. Jenkins)	1
U33 (T. Moore)	1
U33 (C. Nguyen)	1
U33 (T. Park)	1
U33 (H. Roth)	1
U33 (M. Trinh)	1
U40	1

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1 October 1991	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Engineering of Massively Interconnected Computer Systems	5. FUNDING NUMBERS PE - 0602234N PR - RS34P11 TA - Task 4
6. AUTHOR(S) Michael Jenkins, Charles Yeh, and Steven Howell	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Surface Warfare Center (Code U33) 10901 New Hampshire Avenue Silver Spring, MD 20903-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NAVSWC TR 91-588
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document provides an analysis of design elements whose implementation involves algorithmic tasks and hardware resources which must handle large amounts of data or which otherwise present communication (date flow) impasses to the timely (or functional) fulfillment of a design. It proposes a methodology for evaluating candidate resources and selecting those which reduce communication complexity and meet system requirements. The techniques of this methodology should integrate with other steps in the design process so that necessary information is not ignored or lost. They should also be easily automated, since the number of combinations involved may be intractable if attempted by hand. As an indication of future research, it presents background information on resources used in the design of such systems and addresses the implications of introducing physical constraints on these design elements.

14. SUBJECT TERMS Massively Interconnected Computer Systems; Parallel and Distributed Systems; Massive Network Communication; Algorithm and Network Architectures	15. NUMBER OF PAGES 39
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR